PCIe AER support
for
Guest VM user-mode drivers

High level design
&
Performance

Modification History

Revision	Date	Originator	Comments
0.1	12/03/10	Etienne Martineau	Initial draft

Table of Contents

1 Abstract

When a devices is driven by user-space (like in the cases of KVM's device assignment) it's essential to provide proper PCI error handling support to the corresponding driver.

2 Introduction

This document is a high level description of the AER reporting scheme for assigned devices for both user mode and kernel mode driver under qemu-kvm virtual machine.

3 AER overview

Error handling happens in two phases; (1) Reporting, (2) Recovery.

3.1 Phase 1, Reporting (.error_detected callback)

The purpose of the reporting phase is to A) notify the driver so that it can stop all I/O access to the devices quickly and B) provide an exit path to I/O spinloops or inconsistent internal state.

Kernel assume that once the corresponding '.error_detected' callback return, the driver is in quiescent state. Ideally, interruptions to the device should also be disabled.

If the driver doesn't respond to the '.error_detected' callback in a timely fashion, it may result in a scenario where an accumulation of posted writes choke the PCIe switch and eventually the CPU complex.

3.2 Phase 2, Recovery(.mmio_enabled .link_reset .slot_reset .resume)

The purpose of the recovery phase is to provide a mechanism for the driver to 'reconnect' with the device in a sequenced fashion. The times it takes to perform the recovery process usually doesn't impact the host.

3.3 Reporting and recovery in user-space

Both error reporting and error recovery phases are driven by a state machine where the next state depends on the returned value of the current callback. Depending on the state that state machine is in, the host will perform various operation such as 'slot reset' or 'link reset'.

Because of such the only way to maintain consistency between the device and it's corresponding driver in user-space is to perform both phases in lockstep with the host.

3.4 Considerations; Error handling in user-space?

Because error handling has system wide impact, a policy is required when delegating the error handling task to user-space.

When the 'policy' is set to 'paranoid', the host kernel doesn't rely on user-space ever therefore as soon as an error is detected related to an assigned device, the corresponding VM is terminated.

When the 'policy' is set to 'strict', the entire reporting and recovery phases must succeed in lock step. Timeout in the lock step sequence terminate the corresponding VM.

The 'lazy' mode is similar to the 'strict' mode except that if a timeout happen in the lock step sequence, the host kernel will take a default path and keep the process up and running.

For performance reason, the 'lazy' mode is interesting because it may very well operate in an asynchronous fashion without the expensive lock step mode. Obviously, the driver's expected recovery sequence _must_ match the default state machine sequence on the host.

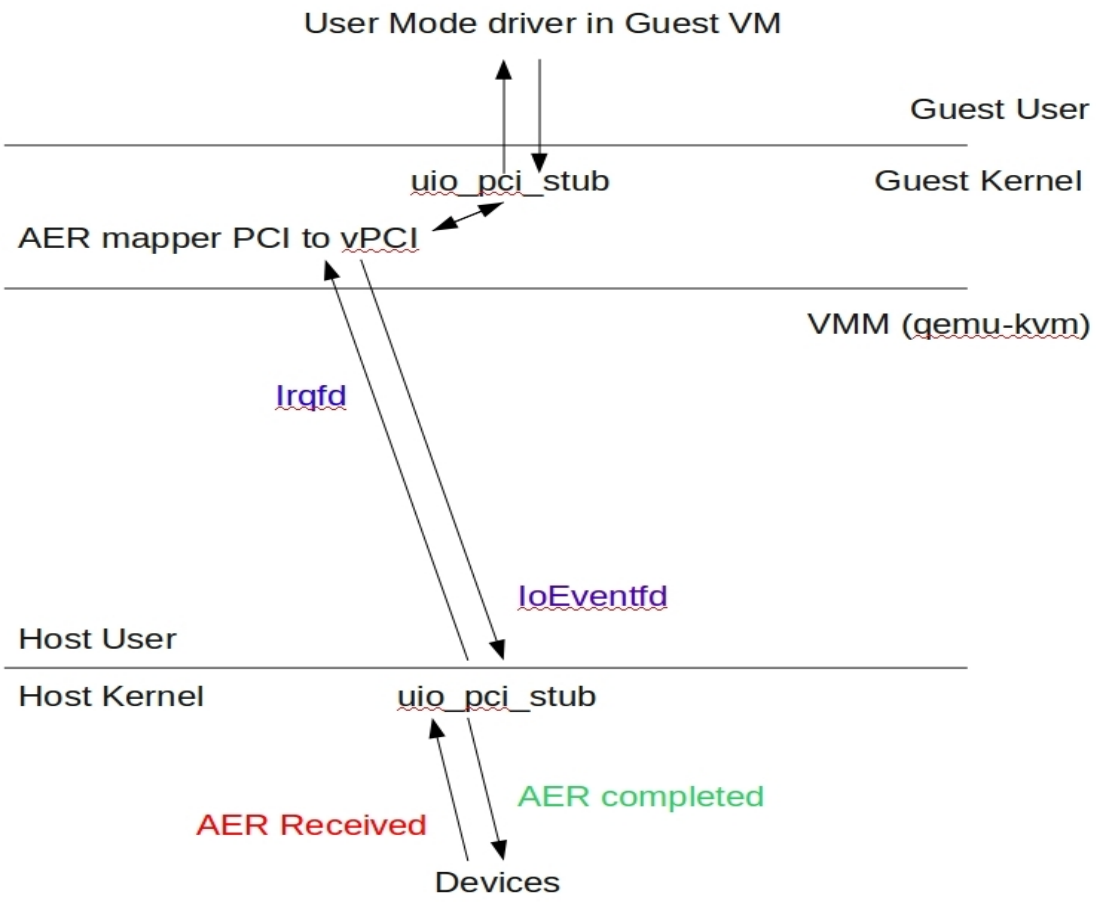
4 Implementation

The PCI error stub driver is implemented on top of the UIO framework. It doesn't require any core kernel patches.

PCI errors are reported to user-space by signaling on an 'eventfd'. Error codes and error results are exposed under a 'logical' BAR that is mmap'able. User-space acknowledge the kernel by using another eventfd which internally is consumed by the PCI error stub driver in a similar fashion than KVM's 'irqfd'.

The UIO implementation accepts 32 bit writes to the device file. This interface is used to multiplex the passing of both eventfd to the PCI error stub driver.

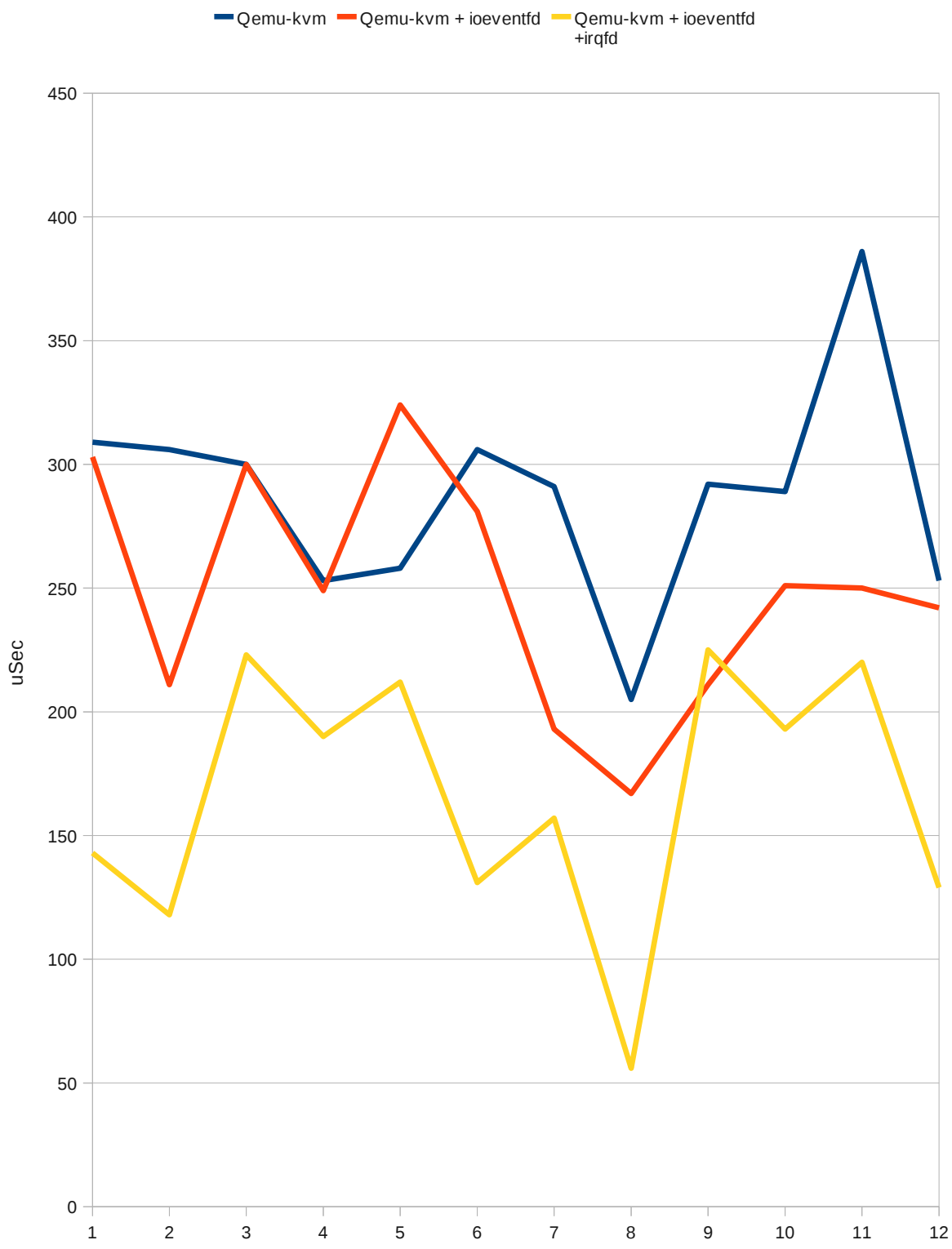
The guest-visible interface is define as a simple QEMU device model. The corresponding kernel instance is called the 'aer-mapper' driver and act as a dispatching agent when AER are received.



4.1 Performance

From performance's perspective, the error signaling path reach the guest directly _without_ any intervention from the VMM (qemu-kvm) using KVM's irqfd. Since both error code and error result are mmap directly, there is also no impact on the guest. Moreover the guest acknowledge directly on an 'ioeventfd' which also bypass entirely the VMM layer.

The following graph illustrate the performance gain when using stock VMM (qemu-kvm) heavy weight exit compare to using 'ioeventfd' only and compare to using 'ioeventfd + irqfd'. Performance improvement shows a 100% gain from stock VMM to 'ioeventfd + irqfd' !


Userspace AER response time
Qemu-kvm
Qemu-kvm + ioeventfd
Qemu-kvm + ioeventfd +irqfd
uSec
450
400
350
300
250
200
150
100
50
0
1
2
3
4
5
6
7
8
9
10
11
12